Efficient Computation of Convolution of Huge Images

David Svoboda

Centre for Biomedical Image Analysis
Faculty of Informatics, Masaryk University, Brno, Czech Republic
svoboda@fi.muni.cz

Abstract. In image processing, convolution is a frequently used operation. It is an important tool for performing basic image enhancement as well as sophisticated analysis. Naturally, due to its necessity and still continually increasing size of processed image data there is a great demand for its efficient implementation. The fact is that the slowest algorithms (that cannot be practically used) implementing the convolution are capable of handling the data of arbitrary dimension and size. On the other hand, the fastest algorithms have huge memory requirements and hence impose image size limits. Regarding the convolution of huge images, which might be the subtask of some more sophisticated algorithm, fast and correct solution is essential. In this paper, we propose a fast algorithm implementing exact computation of the shift invariant convolution over huge multi-dimensional image data.

Keywords: Convolution, Fast Fourier Transform, Divide-et-Impera.

1 Introduction

Convolution is a very important mathematical tool in the field of image processing. It is employed in edge detection [1], correlation [2], optical flow [3], deconvolution [4], simulation [5], etc. Each time the convolution is called plenty of primitive instructions (addition and multiplication) have to be computed. As the amount of processed image data still raises, there is considerable request for fast manipulation with huge image data. This means that the current image processing algorithms have to be capable of handling large blocks of image data in short time. As a lot of image processing methods is based on convolution, we will focus on this mathematical tool and its modifications that bring some acceleration or memory saving.

The basic convolution algorithm traces the individual pixel positions in the input image. In each position, it evaluates inner product of current pixel neighbourhood and flipped kernel. Although the time complexity of the algorithms based on this approach is polynomial [6] this solution is very slow. This is true namely for large kernels. There exist some improvements that guarantee lower complexity, however always with some limitations.

G. Maino and G.L. Foresti (Eds.): ICIAP 2011, Part I, LNCS 6978, pp. 453–462, 2011.

Separable convolution. The higher dimensional convolution with so called *separable* [7] kernels can be simply decomposed into several lower dimensional (cheaper) convolutions. Gaussian, DoG, and Sobel [1] are the representatives of such group of kernels. However, the deconvolution or template matching algorithms based on correlation methods [2] typically use general kernels, which cannot be characterized by special properties like separability.

Recursive filtering. The convolution is a process where the inner product, whose size corresponds to kernel size, is computed again and again in each individual pixel position. One of the vectors, that enter this operation, is always the same. Hence, we can evaluate the whole inner product only in one position while the neighbouring position can be computed as a slightly modified difference with respect to the first position. Analogously, the same is valid for all the following positions. The computation of the convolution using this approach is called *recursive filtering* [7]. Also this method has its drawbacks. The conversion of general convolution kernel into its recursive version is a nontrivial task. Moreover, the recursive filtering often suffers from inaccuracy and instability [8].

Fast convolution. While the convolution in time domain performs an inner product in each pixel position, all we have to do in Fourier domain is point-wise multiplication. Due to this convolution property we can evaluate the convolution in time $O(N \log N)$. This approach is known as a *fast convolution* [9]. Another advantage is that no restrictions are imposed on the kernel. Unfortunately, the excessive space requirements make this approach not very popular.

In this paper, we designed a fast algorithm capable of performing the convolution over huge image data. We considered both the image and the kernel of large size and the dimensions up to 3D. For the simplicity, all the statements will be explained for 1D space. The extension to higher dimensions will be either straightforward or we will explicitly focus on it. We did not impose any restrictions on the convolution kernel, i.e. the kernel neither was separable nor could be simply represented as a recursive filter.

2 Problem Analysis

In the following, we will focus on the efficient implementation that does not impose any special conditions on convolution kernels. Hence, the fast convolution will be under the scope. We will provide the analysis of time and space complexity. Regarding the former one we will focus on the number of complex additions and multiplications needed for the computation of studied algorithms.

Utilizing the convolution theorem and the fast Fourier transform the 1D convolution of two signals f and g requires

$$(M+N)\left[\frac{9}{2}\log_2(M+N)+1\right] \tag{1}$$

steps. Here $\texttt{size}(f) = M$, $\texttt{size}(g) = N$, and the term $(M + N)$ means that the processed image f was zero padded[1] to prevent the overlap effect caused by circular convolution. The kernel was modified in the same way. Another advantage of using Fourier transform stems from its separability. When convolving two 3D images f^{3d} and g^{3d}, where $\texttt{size}(f^{3d}) = M \times M \times M$ and $\texttt{size}(g^{3d}) = N \times N \times N$, we need only

$$(M+N)^3 \left[\frac{9}{2} \log_2 (M+N)^3 + 1 \right] \tag{2}$$

steps in total. Up to now, this method seemed to be optimal. Before we proceed, let us have a look at the space complexity of this approach.

If we do not take into account buffers for the input/output images and serialize both Fourier transforms, we need space for two equally aligned Fourier images and some negligible Fourier transform workspace. In total, it is

$$(M + N) \cdot C \tag{3}$$

bytes, where $(M+N)$ is a size of one padded image and C is a constant dependent on the required algorithm precision (single, double or long double). If the double precision is required, for example, then $C = 2 \cdot sizeof(double)$, which corresponds to two Fourier images used by real-valued FFT. In 3D case, when $\texttt{size}(f^{3d}) = M \times M \times M$ and $\texttt{size}(g^{3d}) = N \times N \times N$ the space needed by the aligned image data is proportionally higher: $(M + N)^3 \cdot C$ bytes.

For better insight, let us consider the complex simulation process, in which the convolution of two relatively small images 500×500×224 voxels and 128×128×100 voxels is called. In this example, the filtered image was a fraction of some larger 3D microscopic image and the kernel was an empirically measured point spread function (PSF). The convolution was one of image processing algorithm called from our simulation toolbox[2]. When this convolution was performed in double precision on Intel Xeon QuadCore 2.83 GHz computer it took 41 seconds using fast convolution while it lasted cca for 4 days when asked for the computation based on the basic approach. Although the FFT based approach seemed to be very promising, it disappointed regarding the memory requirements. Here are the reasons:

- For the computation of fast convolution, both the image and the kernel must be aligned (typically padded) to the same size.
- If not padded with sufficiently large zero area, the result may differ from that of basic approach due to the periodicity of discrete FT.
- In order to minimize the numerical inaccuracy, FFT over large memory blocks must be computed at least in double precision. In the example above, 2 GB of physical memory was required.

It is clear that using this method the memory overhead is very high. In the following, we will try to reduce the memory requirements while keeping the efficiency of the whole convolution process.

[1] The size of padded image should be exactly $(M + N - 1)$. For the sake of simplicity, we reduced this term to $(M + N)$ as we suppose $M \gg 1$ and $N \gg 1$.

[2] CytoPacq – a simulation toolbox: http://cbia.fi.muni.cz/simulator/

3 Method

Keeping in mind that due to the lack of available memory, direct computation of fast convolution is not realizable using common computers we will try to split the whole task into several subtasks. This means that the image and kernel will be split into smaller pieces, so called *tiles* that need not be of the same size.

3.1 Image Tiling

Splitting the convolved image f into smaller tiles $f_1, f_2, \ldots, f_m$, then performing m smaller convolutions $f_i \otimes g, i = \{1, \ldots, m\}$ and finally merging the results together with discarding the overlaps is a well known algorithm in digital signal processing. The implementation is commonly known as the *overlap-save method* [9].

Let us inspect the memory requirements for this approach. Again, let $\texttt{size}(f) = M$ and $\texttt{size}(g) = N$. As the filtered image f is split into m pieces, the respective memory requirements are lowered to

$$\left(\frac{M}{m} + N\right) \cdot C \tag{4}$$

bytes (compare to Eq. (3)). Hence, a slight improvement was reached. Concerning the time complexity, the fast convolution of two 1D signals of length M and N required $(M+N)(\frac{9}{2}\log_2(M+N)+1)$ steps. If the image is split into m tiles, we need to perform

$$(M+mN)\left[\frac{9}{2}\log_2\left(\frac{M}{m}+N\right)+1\right] \tag{5}$$

steps in total. If there is no division ($m=1$) we get the time complexity of the fast approach. If the division is total ($m=M$) we get even worse complexity than the basic convolution has. The higher the level of splitting is required the worse the complexity is. Therefore, we can conclude that splitting only the image into tiles does not help.

3.2 Kernel Tiling

From the previous text, we recognize that splitting only the image f might be inefficient. It may even happen that the kernel g is so large that splitting of only the image f does not reduce the memory requirements sufficiently. As the convolution belongs to commutative operators one could recommend swapping the image and the kernel. This may help, namely when the image is small and the kernel is very large. As soon as the image and the kernel are swapped, we can simply apply the overlap-save method. However, this approach fails when both the image and the kernel are too large.

Let us decompose the kernel g as well. Keeping in mind that the image f has already been decomposed into m tiles, we can focus on the manipulation with just one image tile $f_i, i \in \{1, \ldots, m\}$. For the evaluation of convolution of the

```
 1: (f, g) ← (input image, kernel)
 2: f → f_1, f_2, ..., f_m {split 'f' into tiles according to overlap-save scheme}
 3: g → g_1, g_2, ..., g_n {split 'g' into tiles according to overlap-add scheme}
 4: h ← 0 {create the output image 'h' and fill it with zeros}
 5: for i = 1 to m do
 6:    for j = 1 to n do
 7:       h_ij ← convolve(f_i, g_j)
                {use fast convolution}
 8:       h_ij ← discard_overruns(h_ij)
                {discard h_ij overruns following overlap-save output rules}
 9:       h ← h + shift(h_ij)
                {add h_ij to h following overlap-add output rules}
10:    end for
11: end for
12: Output ← h
```

Algorithm 1. Divide-et-impera approach applied to the convolution over large image data.

selected image tile f_i and the large kernel g we will employ so called *overlap-add method* [9]. This method splits the kernel g into n pieces, then it performs n smaller convolutions $f_i \otimes g_j, j = \{1, \ldots, n\}$, and finally it adds the partial solutions together preserving the appropriate overruns. This way, we obtain the Algorithm 1. For the detailed description of this algorithm see Figure 1.

3.3 Efficiency

Let us suppose the image f is split into m tiles and kernel g is decomposed into n tiles. The time complexity of the fast convolution $f_i \otimes g_j$ is

$$\left(\frac{M}{m}+\frac{N}{n}\right)\left[\frac{9}{2}\log_2\left(\frac{M}{m}+\frac{N}{n}\right)+1\right]. \tag{6}$$

We have m image tiles and n kernel tiles. In order to perform the complete convolution $f \otimes g$ we have to perform $m \times n$ convolutions (see the nested loops in Algorithm 1) of the individual image and kernel tiles. In total, we have to complete

$$(Mn+Nm)\left[\frac{9}{2}\log_2\left(\frac{M}{m}+\frac{N}{n}\right)+1\right] \tag{7}$$

steps. One can clearly see that without any division ($m = n = 1$) we get the complexity of fast convolution, i.e. the class $O((M+N)\log(M+N))$. For total division ($m = M$ and $n = N$) we obtain basic convolution, i.e. the complexity class $O(MN)$. Concerning the space occupied by our convolution algorithm, we need

$$\left(\frac{M}{m}+\frac{N}{n}\right)\cdot C \tag{8}$$

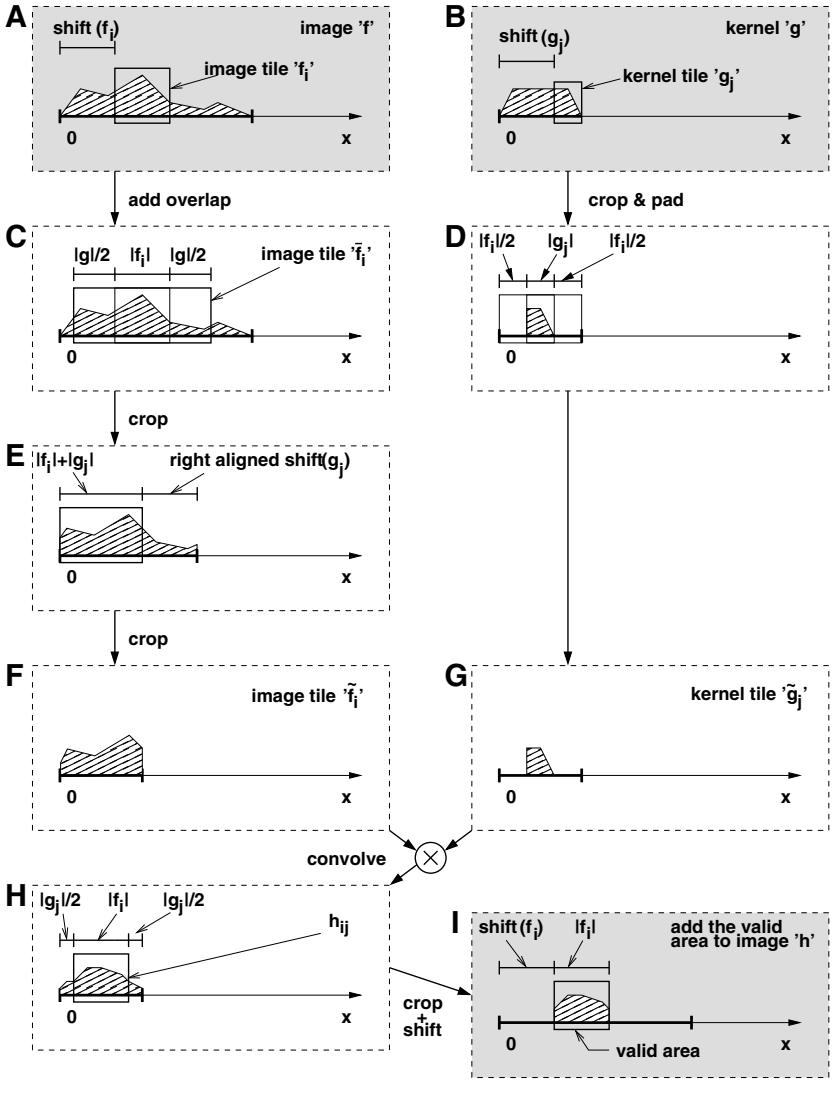

Fig. 1. Convolution of image tile f_i and kernel tile g_j: (A,B) The tiles f_i and g_j are selected. Remember the shift of these tiles with respect to the image they belong to. These shifts will be used later. (C) The tile f_i is extended and cropped. In this way, we obtain $\overline{f}_i$. (D) The tile g_j is cropped (redundant zeros are removed) and padded with zeros in order to get $\widetilde{g}_j$. (E) The area of size $\texttt{size}(f_i)+\texttt{size}(g_j)$ with the distance $\texttt{shift}(g_j)$ from the right border is cropped. In this way, we get $\widetilde{f}_i$. (F,G) The image buffers are aligned to the same size. They are ready for convolution. (H) The convolution is performed. (I) The solution h_{ij} is cut out and shifted to the correct position. Finally, it is added to the output image h. Take note of the light-gray background of some frames. In this way, we distinguish between input/output images (gray frames) and intermediate results (white frames).

bytes, where C is again the precision dependent constant and m, n are the levels of division of image f and kernel g, respectively.

All the previous statements are related only to 1D signal. Provided both image and kernel are D-dimensional cubes and the tiling proces is regular, we can combine Eq. (2) and Eq. (7) in order to get:

$$(Mn+Nm)^D \left[\frac{9}{2}\log_2\left(\frac{M}{m}+\frac{N}{n}\right)^D + 1\right] \tag{9}$$

This statement can be further generalized, i.e. the image and the kernel do not have be in the shape of cube and the tiling does not have to be identical in all the axes. It can be simply derived from the separability of multidimensional Fourier transform, which guarantees that the time complexity of the higher dimensional Fourier transform depends on the amount of processed voxels only. There is no difference in the time complexity if the higher-dimensional image is elongated or in the shape of cube.

4 Results

The proposed algorithm was compared to other freely available implementations of fast convolution – see Figure 2. For the computation of Fourier transform all three packages used FFTW library [10]. It can be clearly seen that the speed of our new approach is comparable with those implemented in the most common image processing toolkits and mostly it outperforms them. Unlike other toolkits, if the computer has less memory than required our new approach does not fail. It splits the task into subtasks and delegates the computing to the loop that successively executes the individual subtasks.

In the previous section, an algorithm of splitting the image f into m tiles and the kernel g into n tiles was proposed. Now we will answer the question regarding the optimal way of splitting the image and the kernel. We still assume that $\texttt{size}(f) = M$ and $\texttt{size}(g) = N$. As M and N are constants let us focus only on the relationship between m and n. Let us define the following substitution: $x = \frac{M}{m}$ and $y = \frac{N}{n}$. Here x and y stand for the sizes of the image and the kernel tiles, respectively. Applying this substitution to Eq. (7) and simplifying, we get

$$MN\left(\frac{1}{x}+\frac{1}{y}\right)\left[\frac{9}{2}\log_2(x+y)+1\right] \tag{10}$$

The plot of this function is depicted in Figure 3. The minimum of this function is reached if and only if $x=y$ and both variables x and y are maximized, i.e. the image and the kernel tiles should be of the same size and they should be as large as possible. In order to reach the optimal solution, the size of the tile should be the power of small primes [11]. In this sense, it is recommended to fulfill both criteria put on the tile size: the maximality (as stated above) and the capability of simple decomposition into small primes.

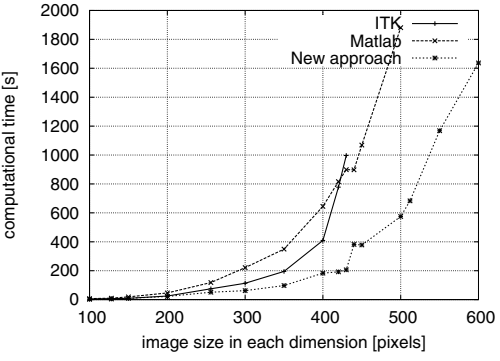

Fig. 2. A graph offering the comparison of the most common implementations of convolution and the new approach. Evaluated over two 3D images of identical size on Intel Xeon QuadCore 2.83 GHz computer with 32 GB RAM. Take note, that ITK and Matlab plots finish earlier as the computation for the images of large dimensions failed due to the lack of memory.

5 Application

Convolution is a core part of many image processing algorithms. In our group we employ the convolution in the process of simulation of fluorescence microscopy images [12]. The simulation process usually consists of three consecutive stages: generation of digital phantoms [13], simulation of optical system, and simulation of acquisition device. We use the convolution in the first two stages:

5.1 Generation of Digital Phantoms

A digital phantom is an image of an estimated model of studied object. As soon as the model is properly defined, a generation of the phantom is straightforward. However, if we want to generate more phantoms we have to solve the problem of collisions as it is reasonable to require the generation of non-overlapping objects only. For this purpose we utilized a correlation. Here, the input image is usually a large image with previously generated phantoms inside and the kernel is a smaller image with a newly generated phantom that should be added to the input image. The output correlated image contains zeros in the positions, where the new phantom will not overlap any of the already existing objects. As the only difference between correlation and convolution stems in the flipping of convolution kernel, the algorithm responsible for the correct placement of newly generated phantoms was substituted with the convolution with flipped kernel. The volume of the image to be correlated was set to the size $55\times55\times12$ microns which corresponded to $1024\times1024\times224$ voxels for the selected optical system[3].

[3] Optical system configuration: microscope Zeiss 200M, objective Zeiss Plan-Apochromat (100×/1.40 Oil), confocal unit Yokogawa CSU-10, and camera Andor iXon DU888E (EM CCD 1024×1024 with pixel size 13 microns)

The kernel size corresponded to the bounding box volume of simulated object. In our case its was in average $10 \times 10 \times 10$ microns ($= 187 \times 187 \times 187$ voxels). In this example, the correlation memory requirements rose to 9.6 GB.

As long as we generated large complex cell structures we avoided the lack of memory because the computer we used was equipped with 32 GB RAM. The problem came out when the multiple spots within a large area were under the scope. Due to low default image resolution, the size of each spot was almost the same as one voxel. However, while the spot is expected to be of spherical shape, the voxel is always rectangular. The remedy was to sufficiently up-sample the image and the kernel. When we asked for n-tuple subpixel precision the size of both image and kernel rose n-times along each axis! The excessive memory requirements made the standard fast convolution approach unusable. For this purpose we used our new approach that can handle the data of any size.

5.2 Simulation of Optical System

The simulation of optical system is a next step that directly follows up on the generation of digital phantoms. The optical system is characterized by its PSF, that describes how the infinitely small impulse is transformed when passing through this system. This transform is usually modelled as a convolution, where the image to be convolved is the image containing digital phantoms and the convolution kernel is represented by the PSF. In this task we used the same optical system configuration and the same input image as in the example above. The kernel size, which was an empirically measured PSF (prepared by SVI Huygens® Pro software), was fixed to the size of $128 \times 128 \times 100$ voxels. The memory requirements for the convolution started at about 6.8 GB and they accordingly rose when asked for higher level of subpixel precision. Therefore, we again used our new approach capable of working with large image data.

6 Conclusion

We designed a fast algorithm for the computation of the convolution. While all the current methods impose some restrictions on the input data (size of the data, separable kernel, size of kernel and image should be powers of small primes), we do not require any. Prior to the computation, our algorithm verifies the amount of available memory. If there is not enough memory the principle of *divide-et-impera* is applied. Therefore, the unsolvable huge problem is split into several smaller subproblems. In our approach we split the input data (the image and the kernel). The efficiency of the computation highly depends on the level of division. If no division is realized, the time complexity is equal to the complexity of the fast convolution. On the other hand, if we were forced to split the image and the kernel into individual pixels the time complexity of the algorithm would belong to the same class as the basic approach. Following the graph in Fig. 3 one can clearly see that all the other cases belong to the complexity classes in between. In practice, we apply only a small level of division which leads nearly to the optimal

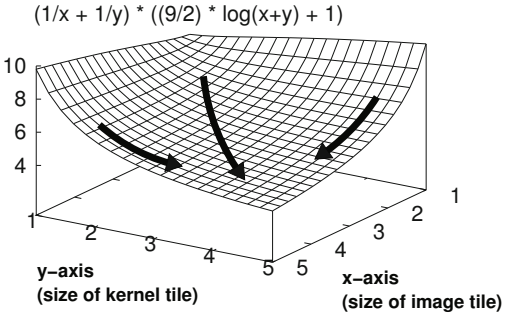

Fig. 3. A graph of a function that represents the time complexity of tiled convolution. The evident minimum occurs in the location, where both variables (size of the tiles) are maximized and equal at the same time.

solution (fast convolution). The implementation of this convolution algorithm is part of image processing library which is under GNU GPL and is available at: `http://cbia.fi.muni.cz/projects/i3dlibs.html`. The algorithm has been successfully used in the simulation toolbox developed in our group.

Acknowledgment. This work was supported by the Ministry of Education of the Czech Rep. (Projects No. MSM0021622419, No. LC535 and No. 2B06052).

References

1. Parker, J.R.: Algorithms for image processing and computer vision. Wiley, Chichester (1996)
2. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Prentice-Hall, Englewood Cliffs (2002)
3. Fleet, D.J., Jepson, A.D.: Computation of component image velocity from local phase information. Int. J. Comput. Vision 5(1), 77–104 (1990)
4. Verveer, P.J.: Computational and optical methods for improving resolution and signal quality in fluorescence microscopy, PhD Thesis. Delft Technical Univ. (1998)
5. Jensen, J.A., Munk, P.: Computer phantoms for simulating ultrasound B-mode and cfm images. Acoustical Imaging 23, 75–80 (1997)
6. Pratt, W.K.: Digital Image Processing. Wiley, Chichester (1991)
7. Jähne, B.: Digital Image Processing, 6th edn. Springer, Heidelberg (1997)
8. Smith, S.W.: Digital Signal Processing. Newnes (2002)
9. Jan, J.: Digital Signal Filtering, Analysis and Restoration. Telecommunications Series. INSPEC, Inc. (2000) ISBN: 0852967608
10. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proceedings of the IEEE 93(2), 216–231 (2005); Special issue on Program Generation, Optimization, and Platform Adaptation
11. Heideman, M., Johnson, D., Burrus, C.: Gauss and the history of the fast fourier transform. IEEE ASSP Magazine 1(4), 14–21 (1984) ISSN: 0740-7467
12. Svoboda, D., Kozubek, M., Stejskal, S.: Generation of digital phantoms of cell nuclei and simulation of image formation in 3d image cytometry. Cytometry Part A 75A(6), 494–509 (2009)
13. Rexilius, J., Hahn, H.K., Bourquain, H., Peitgen, H.-O.: Ground Truth in MS Lesion Volumetry – A Phantom Study. In: Ellis, R.E., Peters, T.M. (eds.) MICCAI 2003. LNCS, vol. 2879, pp. 546–553. Springer, Heidelberg (2003)